CHAPTER 2

NoSQL

"NoSQL is a new way of designing Internet-scale database solutions. It is not a product or technology but a term that defines a set of database technologies that are not based on the traditional RDBMS principles."

In this chapter, we will cover the definition and basics of NoSQL. We will introduce you to the CAP theorem and will talk about the NRW notations. We will compare the ACID and BASE approaches and finally conclude the chapter by comparing NoSQL and SQL database technologies.

SQL

The idea of RDBMS was borne from E.F. Codd's 1970 whitepaper titled "A relational model of data for large shared data banks." The language used to query RDBMS systems is SQL (Sequel Query Language).

RDBMS systems are well suited for structured data held in columns and rows, which can be queried using SQL. The RDBMS systems are based on the concept of ACID transactions. ACID stands for Atomic, Consistent, Isolated, and Durable, where

- **Atomic** implies either all changes of a transaction are applied completely or not applied at all.
- **Consistent** means the data is in a consistent state after the transaction is applied. This means after a transaction is committed, the queries fetching a particular data will see the same result.
- **Isolated** means the transactions that are applied to the same set of data are independent of each other. Thus, one transaction will not interfere with another transaction.
- **Durable** means the changes are permanent in the system and will not be lost in case of any failures.

NoSQL

NoSQL is a term used to refer to non-relational databases. Thus, it encompasses majority of the data stores that are not based on the conventional RDBMS principles and are used for handling large data sets on an Internet scale.

Big data, as discussed in the previous chapter, is posing challenges to the traditional ways of storing and processing data, such as the RDBMS systems. As a result, we see the rise of NoSQL databases, which are designed to process this huge amount and variety of data within time and cost constraints.

S.G. Edward and N. Sabharwal, *Practical MongoDB,* DOI 10.1007/978-1-4842-0647-8_2

Thus NoSQL databases evolved from the need to handle big data; traditional RDBMS technologies could not provide adequate solutions. Figure 2-1 shows the rise of un/semi-structured data over the years as compared to structured data.

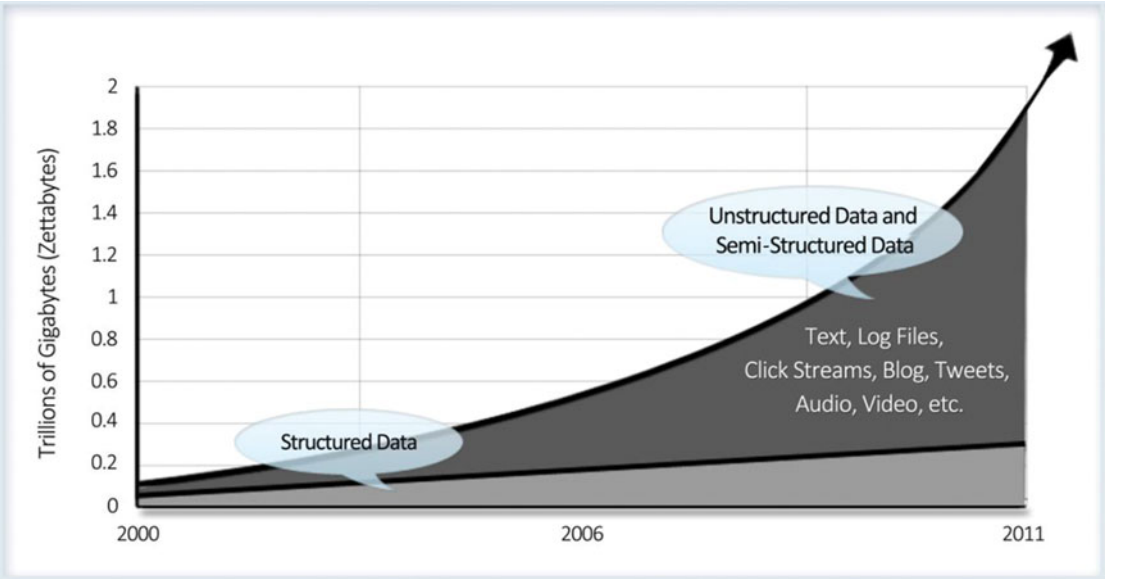

Figure 2-1. *Structured vs. un/Semi-Structured data*

Some examples of big data use cases that are a good fit for NoSQL databases are the following:

- **Social Network Graph**: Who is connected to whom? Whose post should be visible on the user's wall or homepage on a social network site?
- **Search and Retrieve**: Search all relevant pages with a particular keyword ranked by the number of times a keyword appears on a page.

Definition

NoSQL doesn't have a formal definition. It represents a form of persistence/data storage mechanism that is fundamentally different from RDBMS. But if pushed to define NoSQL, here it is: NoSQL is an umbrella term for data stores that don't follow the RDBMS principles.

■ **Note** The term was used initially to mean "do not use SQL if you want to scale." Later this was redefined to "not only SQL," which means that in addition to SQL other complimentary database solutions exist.

A Brief History of NoSQL

In 1998, Carlo Strozzi coined the term *NoSQL*. He used this term to identify his database because the database didn't have a SQL interface. The term resurfaced in early 2009 when Eric Evans (a Rackspace employee) used this term in an event on open source distributed databases to refer to distributed databases that were non-relational and did not follow the ACID features of relational databases.

ACID vs. BASE

In the introduction, we mentioned that the traditional RDBMS applications have focused on ACID transactions. Howsoever essential these qualities may seem, they are quite incompatible with availability and performance requirements for applications of a Web scale.

Let's say, for example, that you have a company like OLX, which sells products such as unused household goods (old furniture, vehicles, etc.) and uses RDBMS as its database. Let's consider two scenarios.

First scenario: Let's look at an e-commerce shopping site where a user is buying a product. During the transaction, the user locks a part of database, the inventory, and every other user must wait until the user who has put the lock completes the transaction.

Second scenario: The application might end up using cached data or even unlocked records, resulting in inconsistency. In this case, two users might end up buying the product when the inventory actually was zero.

The system may become slow, impacting scalability and user experience.

In contrary to the ACID approach of traditional RDBMS systems, NoSQL solves the problem using an approach popularly called as BASE. Before explaining BASE, let's explore the concept of the CAP theorem.

CAP Theorem (Brewer's Theorem)

Eric Brewer outlined the CAP theorem in 2000. This is an important concept that needs to be well understood by developers and architects dealing with distributed databases. The theorem states that when designing an application in a distributed environment there are three basic requirements that exist, namely consistency, availability, and partition tolerance.

- **Consistency** means that the data remains consistent after any operation is performed that changes the data, and that all users or clients accessing the application see the same updated data.
- **Availability** means that the system is always available.
- **Partition Tolerance** means that the system will continue to function even if it is partitioned into groups of servers that are not able to communicate with one another.

The CAP theorem states that at any point in time a distributed system can fulfil only two of the above three guarantees (Figure 2-2).

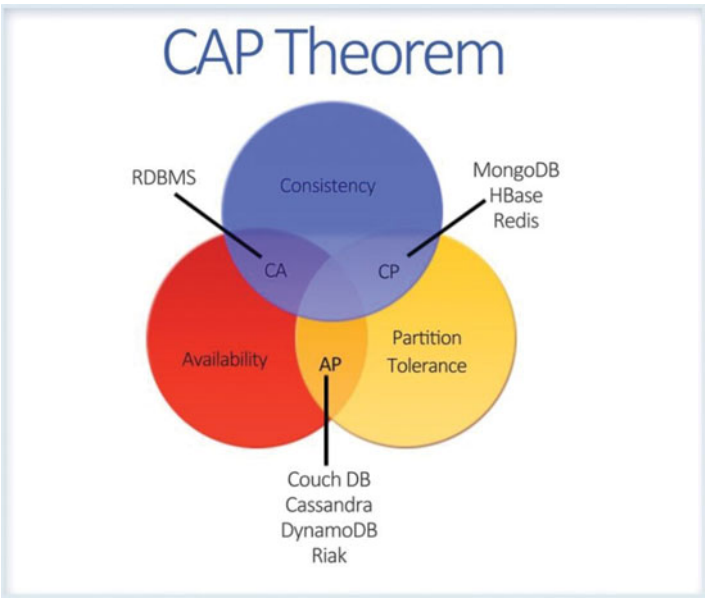

Figure 2-2. *CAP Theorem*

The BASE

Eric Brewer coined the BASE acronym. BASE can be explained as

- **Basically Available** means the system will be available in terms of the CAP theorem.
- **Soft state** indicates that even if no input is provided to the system, the state will change over time. This is in accordance to eventual consistency.
- **Eventual consistency** means the system will attain consistency in the long run, provided no input is sent to the system during that time.

Hence BASE is in contrast with the RDBMS ACID transactions.

You have seen that NoSQL databases are eventually consistent but the eventual consistency implementation may vary across different NoSQL databases.

NRW is the notation used to describe how the eventual consistency model is implemented across NoSQL databases where

- **N** is the number of data copies that the database has maintained.
- **R** is the number of copies that an application needs to refer to before returning a read request's output.
- **W** is the number of data copies that need to be written to before a write operation is marked as completed successfully.

Using these notation configurations, the databases implement the model of eventual consistency.

Consistency can be implemented at both read and write operation levels.

Write Operations

N=W implies that the write operation will update all data copies before returning the control to the client and marking the write operation as successful. This is similar to how the traditional RDBMS databases work when implementing synchronous replication. This setting will slow down the write performance.

If write performance is a concern, which means you want the writes to be happening fast, you can set W=1, R=N. This implies that the write will just update any one copy and mark the write as successful, but whenever the user issues a read request, it will read all the copies to return the result. If either of the copies is not updated, it will ensure the same is updated, and then only the read will be successful. This implementation will slow down the read performance.

Hence most NoSQL implementations use N>W>1. This implies that greater than one node needs to be updated successfully; however, not all nodes need to be updated at the same time.

Read Operations

If R is set to 1, the read operation will read any data copy, which can be outdated. If R>1, more than one copy is read, and it will read most recent value. However, this can slow down the read operation.

Using N<W+R always ensures that a read operation retrieves the latest value. This is because the number of written copies and read copies are always greater than the actual number of copies, ensuring that at least one read copy has the latest version. This is **quorum assembly**.

Table 2-1 compares ACID vs. BASE.

Table 2-1. *ACID vs. BASE*

ACID	BASE
Atomicity	Basically Available
Consistency	Eventually Consistency
Isolation	Soft State
Durable	

NoSQL Advantages and Disadvantages

In this section, you will look at the advantages and disadvantages of NoSQL databases.

Advantages of NoSQL

Let's talk about the advantages of NoSQL databases.

- **High scalability**: This scaling up approach fails when the transaction rates and fast response requirements increase. In contrast to this, the new generation of NoSQL databases is designed to scale out (i.e. to expand horizontally using low-end commodity servers).

- **Manageability and administration**: NoSQL databases are designed to mostly work with automated repairs, distributed data, and simpler data models, leading to low manageability and administration.
- **Low cost**: NoSQL databases are typically designed to work with a cluster of cheap commodity servers, enabling the users to store and process more data at a low cost.
- **Flexible data models**: NoSQL databases have a very flexible data model, enabling them to work with any type of data; they don't comply with the rigid RDBMS data models. As a result, any application changes that involve updating the database schema can be easily implemented.

Disadvantages of NoSQL

In addition to the above mentioned advantages, there are many impediments that you need to be aware of before you start developing applications using these platforms.

- **Maturity**: Most NoSQL databases are pre-production versions with key features that are still to be implemented. Thus, when deciding on a NoSQL database, you should analyze the product properly to ensure the features are fully implemented and not still on the To-do list.
- **Support**: Support is one limitation that you need to consider. Most NoSQL databases are from start-ups which were open sourced. As a result, support is very minimal as compared to the enterprise software companies and may not have global reach or support resources.
- **Limited Query Capabilities**: Since NoSQL databases are generally developed to meet the scaling requirement of the web-scale applications, they provide limited querying capabilities. A simple querying requirement may involve significant programming expertise.
- **Administration**: Although NoSQL is designed to provide a no-admin solution, it still requires skill and effort for installing and maintaining the solution.
- **Expertise**: Since NoSQL is an evolving area, expertise on the technology is limited in the developer and administrator community.

Although NoSQL is becoming an important part of the database landscape, you need to be aware of the limitations and advantages of the products to make the correct choice of the NoSQL database platform.

SQL vs. NoSQL Databases

Now you know the details regarding NoSQL databases. Although NoSQL is increasingly getting adopted as a database solution, it's not here to replace SQL or RDBMS databases. In this section, you will look at the differences between SQL and NoSQL databases.

Let's do a quick recap of the RDBMS system. RDBMS systems have prevailed for about 30 years, and even now they are the default choice of the solution architect for data storage for an application. If we will list a few of the good points of RDBMS system, first and the foremost is the use of SQL, which is a rich declarative query language used for data processing. It is well understood by users. In addition, the RDBMS system offers ACID support for transactions, which is a must in many sectors, such as banking applications.

However, the biggest drawbacks of the RDBMS system are its difficulty in handling schema changes and scaling issues as data increases. As data increases, the read read/write performance degrades. You face scaling issues with RDBMS systems because they are mostly designed to scale up and not scale out.

In contrast to the SQL RDBMS databases, NoSQL promotes the data stores, which break away from the RDBMS paradigm.

Let's talk about technical scenarios and how they compare in RDBMS vs. NoSQL:

- **Schema flexibility:** This is a must for easy future enhancements and integration with external applications (outbound or inbound).

  RDBMS are quite inflexible in their design. Adding a column is an absolute no-no, especially if the table has some data. The reasons range from default value, indexes, and performance implications. More often than not, you end up creating new tables, and increasing the complexity by introducing relationships across tables.

- **Complex queries:** The traditional designing of the tables leads to developers writing complex JOIN queries, which are not only difficult to implement and maintain but also take substantial database resources to execute.

- **Data update:** Updating data across tables is probably one of the more complex scenarios, especially if they are a part of the transaction. Note that keeping the transaction open for a long duration hampers the performance. You also have to plan for propagating the updates to multiple nodes across the system. And if the system does not support multiple masters or writing to multiple nodes simultaneously, there is a risk of node failure and the entire application moving to read-only mode.

- **Scalability:** Often the only scalability that may be required is for read operations. However, several factors impact this speed as operations grow. Some of the key questions to ask are:

  - What is the time taken to synchronize the data across physical database instances?
  - What is the time taken to synchronize the data across datacenters?
  - What is the bandwidth requirement to synchronize data?
  - Is the data exchanged optimized?
  - What is the latency when any update is synchronized across servers? Typically, the records will be locked during an update.

  NoSQL-based solutions provide answers to most of the challenges listed above.

  Let's now see what NoSQL has to offer against each technical question mentioned above.

- **Schema flexibility:** Column-oriented databases store data as columns as opposed to rows in RDBMS. This allows the flexibility of adding one or more columns as required, on the fly. Similarly, document stores that allow storing semi-structured data are also good options.

- **Complex queries:** NoSQL databases do not have support for relationships or foreign keys. There are no complex queries. There are no JOIN statements.

  Is that a drawback? How does one query across tables?

  It is a functional drawback, definitely. To query across tables, multiple queries must be executed. A database is a shared resource, used across application servers and must not be released from use as quickly as possible. The options involve a combination of simplifying the queries to be executed, caching data, and performing complex operations in the application tier. A lot of databases provide in-built entity-level caching. This means that when a record is accessed, it may be automatically cached transparently by the database. The cache may be in-memory distributed cache for performance and scale.

- **Data update:** Data updating and synchronization across physical instances are difficult engineering problems to solve. Synchronization across nodes within a datacenter has a different set of requirements compared to synchronizing across multiple datacenters. One would want the latency within a couple of milliseconds or tens of milliseconds at the best. NoSQL solutions offer great synchronization options.

  MongoDB, for example, allows concurrent updates across nodes, synchronization with conflict resolution, and eventually, consistency across the datacenters within an acceptable time that would run in few milliseconds. As such, MongoDB has no concept of isolation. Note that now because the complexity of managing the transaction may be moved out of the database, the application will have to do some hard work.

  An example of this is a two-phase commit while implementing transactions (`http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/`).

  A plethora of databases offer multiversion concurrency control (MCC) to achieve transactional consistency.

  Well, as Dan Pritchett (`www.addsimplicity.com/`), Technical Fellow at eBay puts it, eBay.com does not use transactions. Note that PayPal does use transactions.

- **Scalability:** NoSQL solutions provider greater scalability for obvious reasons. A lot of the complexity that is required for transaction-oriented RDBMS does not exist in ACID non-compliant NoSQL databases. Interestingly, since NoSQL does not provide cross-table references and there are no JOIN queries possible, and because you can't write a single query to collate data across multiple tables, one simple and logical solution is to—at times—duplicate the data across tables. In some scenarios, embedding the information within the primary entity—especially in one-to-one mapping cases—may be a great idea.

Table 2-2 compares SQL and NoSQL technologies.

Table 2-2. *SQL vs. NoSQL*

	SQL Databases	NoSQL Databases
Types	All types support SQL standard.	Multiple types exists, such as document stores, key value stores, column databases, etc.
Development History	Developed in 1970.	Developed in 2000s.
Examples	SQL Server, Oracle, MySQL.	MongoDB, HBase, Cassandra.
Data Storage Model	Data is stored in rows and columns in a table, where each column is of a specific type. The tables generally are created on principles of normalization. Joins are used to retrieve data from multiple tables.	The data model depends on the database type. Say data is stored as a key-value pair for key-value stores. In document-based databases, the data is stored as documents. The data model is flexible, in contrast to the rigid table model of the RDBMS.
Schemas	Fixed structure and schema, so any change to schema involves altering the database.	Dynamic schema, new data types, or structures can be accommodated by expanding or altering the current schema. New fields can be added dynamically.
Scalability	Scale up approach is used; this means as the load increases, bigger, expensive servers are bought to accommodate the data.	Scale out approach is used; this means distributing the data load across inexpensive commodity servers.
Supports Transactions	Supports ACID and transactions.	Supports partitioning and availability, and compromises on transactions. Transactions exist at certain level, such as the database level or document level.
Consistency	Strong consistency.	Dependent on the product. Few chose to provide strong consistency whereas few provide eventual consistency.
Support	High level of enterprise support is provided.	Open source model. Support through third parties or companies building the open source products.
Maturity	Have been around for a long time.	Some of them are mature; others are evolving.
Querying Capabilities	Available through easy-to-use GUI interfaces.	Querying may require programming expertise and knowledge. Rather than an UI, focus is on functionality and programming interfaces.
Expertise	Large community of developers who have been leveraging the SQL language and RDBMS concepts to architect and develop applications.	Small community of developers working on these open source tools.

Categories of NoSQL Databases

In this section, you will quickly explore the NoSQL landscape. You will look at the emerging categories of NoSQL databases. Table 2-3 shows a few of the projects in the NoSQL landscape, with the types and the players in each category.

Table 2-3. *NoSQL Categories*

Category	Brief Description	For E.g.
Document-based	Data is stored in form of documents. For instance, {Name="Test User", Address="Address1", Age:8}	MongoDB
XML database	XML is used for storing data.	MarkLogic
Graph databases	Data is stored as node collections. The nodes are connected via edges. A node is comparable to an object in a programming language.	GraphDB
Key-value store	Stores data as key-value pairs.	Cassandra, Redis, memcached

The NoSQL databases are categorized on the basis of how the data is stored. NoSQL mostly follows a horizontal structure because of the need to provide curated information from large volumes, generally in near real-time. They are optimized for insert and retrieve operations on a large scale with built-in capabilities for replication and clustering.

Table 2-4 briefly provides a feature comparison between the various categories of NoSQL databases.

Table 2-4. *Feature Comparison*

Feature	Column-Oriented	Document Store	Key-Value Store	Graph
Table-like schema support (columns)	Yes	No	No	Yes
Complete update/fetch	Yes	Yes	Yes	Yes
Partial update/fetch	Yes	Yes	Yes	No
Query/filter on value	Yes	Yes	No	Yes
Aggregate across rows	Yes	No	No	No
Relationship between entities	No	No	No	Yes
Cross-entity view support	No	Yes	No	No
Batch fetch	Yes	Yes	Yes	Yes
Batch update	Yes	Yes	Yes	No

The important thing when considering a NoSQL project is the feature set you are interested in. When deciding on a NoSQL product, first you need to understand the problem requirements very carefully, and then you should look at other people who have already used the NoSQL product to solve similar problems. Remember that NoSQL is still maturing, so this will enable you to learn from peers and previous deployments, and make better choices.

In addition, you also need to consider the following questions.

- How big is the data that needs to be handled?
- What throughput is acceptable for read and write?
- How is consistency is achieved in the system?
- Does the system need to support high write performance or high read performance?
- How easy is the maintainability and administration?
- What needs to be queried?
- What is the benefit of using NoSQL?

We recommend that you start small but significant, and consider a hybrid approach wherever possible.

Summary

In this chapter, you learned about NoSQL. You should now understand what NoSQL is and how it is different from SQL. You also looked into the various categories of NoSQL.

In the following chapters, you will look into MongoDB, which is a document-based NoSQL database.